*Symbo*libre

Final project report

https://symbolibre.org

Leaders
CHAPPE Nicolas and MICHELLAND Sébastien

Supervisor
CARON Eddy

Members

BARIL Ambroise
BOONE Victor
CASTILLON Antoine
CHAMPSEIX Nicolas
CYFFERS Edwige
DARRIN Maxime
DELAËT Alain
DOMENECH Antoine
DUBUC Théophile
DURAIN Bastien
DÉPRÉS Hugues
FOURNIER Némo
GASSOT Charles
HILLEBRAND Quentin
HOHNADEL Émile
HUMEAU Samuel
HUTIN Gérémy
JOUANS Loïc
KHERRAF Nabil
LEVY Nicolas
LUNEL Corentin
MERCKLÉ Victor
OIJID Nacim
PAVIET-SALOMON Léo
PETITHOMME Jérémy
TAILLANDIER Valentin
WALCH Aymeric

Contents

Part 1

Introduction

This document is the final report of the ***Symbo*****libre** Integrated Project, developed at *ENS de Lyon* in 2018–2019 under the supervision of Eddy Caron, with the support of teachers from the Institut Français de l'Éducation. During the year, the project team developed a prototype graphing calculator centered around free software and the do-it-yourself philosophy.

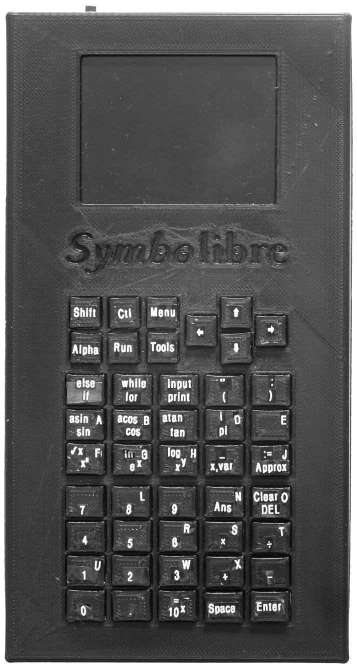


Figure 1.1: First prototype of ***Symbo*****libre**.

1.1 Motivation

Dominant graphing calculator constructors on the French market have put little effort in developing innovating products in recent years. Our favorite and most common Texas Instruments, Casio and HP calculators have not changed a lot in the last 15 years, probably because the activity of the sector does not justify more investments.

This sometimes leads to families of models with identical hardware but different firmwares and thus different prices (TI-76 versus TI-83 Plus, TI-Nspire versus TI-Nspire CAS...). These models are all more expensive than low-end cell phones, but still include significantly less powerful hardware.

Communities have been eager to contribute to the improvement of the software in particular, but the closed-source operating systems and restrictions on native development options block most of these efforts.

The goal of the ***Symbo*****libre** project is to develop a graphing calculator for high school and higher education students, around the free software and do-it-yourself philosophies. The calculator can be assembled from widely-available hardware, and its software is fully open for study, contribution and reuse.

It is an attempt to support modernity and free software in the world of graphing calculators, that has long been dominated by a few constructors with proprietary technologies. As more transparent and efficients products are starting to appear, we hope to play a role in the transition.

1.1.1 Recent evolutions

Graphing calculators is a field that evolves slowly, partly because most manufacturers do not specialize in calculators. In France, their evolution mostly follows legal requirements. Hopefully, the recent introduction of the exam mode has created quite a bit of turmoil, so let's start from here.

This regulation requires all calculators in exams to have a special *examination mode* that temporarily resets the calculator and locks communication and storage functions. This applies to all national exams, including the *baccalauréat*. Although the regulation has still to be enforced, most high-schoolers already had to buy or find a new calculator. This was a significant opportunity for constructors to develop and sell new models.

Yet little effort has been made to improve their products. As a notable illustration, the introduction of Python programming in high-school programs [1] reveals the old technology at work. Casio's port of Micropython is even slower than their native Basic language; TI devised an external module to compensate for a lack of computational power; and despite having a built-in Pyhon application, the Numworks is also struggling with performance. The HP Prime puts up a decent fight, but is already a top-of-the-range model. [2]

The declining technology is also seen in hardware; the most powerful machines work at a 100 MHz frequency with a few Megabytes of RAM. This is barely enough to support non-trivial computer algebra, a feature deemed to be worth at least €100.

1.1.2 Overview of current graphing calculators

TI products Texas Instruments currently maintains two series of graphing calculators.

- TI calculators based on a z80 processor (lots of variants with names starting with TI-81 to TI-86). They offer no symbolic computation features. This series first appeared in 1990 with the TI-81 calculator, but a TI-84 Plus CE (2015) is not that different on the hardware and software levels. The initial retail price is higher though!
- The TI-Nspire, first released in 2007, is their current high-end calculator, with optional computer algebra capabilities (at a higher price). Besides Basic, it supports Lua programming.

Casio products Casio currently has three graphing calculator series.

- The *Graph* series, dating back to the Graph 80 in 1998, up to a series of hardware- and software-compatible products that share variations of the same operating system since 2005. This includes Casio's best-selling machine, the Graph 35+;
- The *Prizm* series, which started in 2011 with the fx-CG 10 and 20 that were color versions of the *Graph* products of the time, and was revisited in 2017 with the popular and elaborate fx-CG 50, misleadingly called Graph 90+E in France.
- And a less-known *Classpad* series with large touch screens, that culminated in 2012 with the expensive Classpad 400.

HP products

HP has several models, but only the HP Prime is ever heard of in France. It ranks as a top-notch calculator and runs Bernard Parisse's well-known Giac computer algebra system. [6] Unfortunately, as every high-end model, it is very costly (about €150) and apart from Giac the environment isn't quite open.

The Numworks calculator

The Numworks calculator was introduced in mid-2017 as a modern and somewhat open calculator. The source code of the operating system Epsilon is available on the web [4] and contributions from the community are accepted. Hardware schematics are also public.

It is unfortunate that their licensing conditions (Creative Commons BY-NC-SA) prevent any reuse of their very specialized work in any project with commercial applications. As for their schematics, no derivative work is allowed. The project falls short of free software philosophy at this point.

1.1.3 Short technical analysis

Figure 1.2 is a simple feature comparison of a few mainstream calculator models and the design of the ***Symbo*****libre** calculator. It is worth noting that the second revision of the HP Prime has very high capabitilies similar to that of our calculator; unfortunately it is near-unobtainable in France. We focus here on the most common models.

Please note that the price of the ***Symbo*****libre** calculator is indicated for a do-it-yourself situation where components are ordered and assembled manually, excluding possible shipping costs.

Calculator	**Graph 35+E**	**NSpire CX CAS**	**Numworks**	***Symbo*****libre**
Free software	No	No	Somewhat	Yes
CAS	Community	NSpire CAS	No	Giac
Powering	AAA batteries	Rechargeable	Rechargeable	Rechargeable
Programming	Basic	Basic	Python	Python
Python	Unofficial	External device	Built-in	Built-in
Exam mode	Yes	Yes	Yes	Yes
Display	Mono 128×64	Color 320×240	Color 320×240	Color 320×240
CPU Freq.	30 MHz	133 MHz	100 MHz	1 GHz
RAM	512 kB	64 MB	256 kB	512 MB
Price	€65-70	€120-180	€80-100	€50-65

Figure 1.2: Comparison of the main features of common calculators.

1.1.4 Transition

Communities have long resented the lack of transparency and control of calculators, leading to projects such as the late LibreCalc [5], a praiseworthy attempt at building a free-hardware, free-software calculator.

With the ***Symbo*****libre** project, we hope to support this transition towards modern, free-minded calculators by setting an example. Our calculator can be built from the inexpensive Rapsberry Pi Zero and a Linux kernel, uses the free computer algebra software Giac, and gives its user full control over the hardware and software.

1.2 Organization

The project was split into seven roughly independent groups, shown in figure 1.3.

No.	Members	Name	Tasks
1	4	Hardware	Build the electronics, case and keyboard.
2	2	Sysadmin	Set up the operating system and the toolchain.
3	4	Math applications	Implement graphical math appliations using Giac.
4	4	Programming	IDE and support for common languages.
5	3	System tools	Communication with computer, main menu.
a	5	Communication	Run the survey, IFÉ partnership, website.
b	6	SageMath	Turing machines implementation in SageMath.

Figure 1.3: Overview of the project's work distribution.

(1) Hardware, led by Némo Fournier The hardware group was assigned the choice and assembly of the components of the calculator. This task is constrained by the do-it-yourself distribution model which requires us to use mainstream and well-documented hardware. This group also designed and printed the calculator case and keyboard shown in Figure 1.1. Its work is detailed in part 2.

(2) System administration, led by Nicolas Chappe The calculator runs a GNU/Linux system built on top of the Gentoo distribution with the specific drivers we need. This group is responsible for compiling and updating these components, as well as the desktop environment and the graphical libraries used by the applications. Its work is presented in part 3.

(3) Math applications, led by Aymeric Walch The actual math work is handled by Giac, including everything from core functions to symbolic calculations to equation solving. This team worked on graphical interfaces to nicely expose these features, with a focus on 2D math rendering, graph plotting, and several aspects of user experience. Its work is shown in part 4.

(4) Programming, led by Maxime Darrin Modern calculators have to support programming, in accordance with high school and higher education programs. The programming group implemented a text editor with some IDE features and support for the common subsets of TI-Basic and Casio Basic. Their results are described in part 4.

(5) System tools, led by Loïc Jouans The system also provides tools to communicate with desktop computers with a custom program for complex tasks, while allowing user to access calculator data via the file browser of the computer. These solutions were set up by the system tools group and are discussed in part 3.

(a) Communication, led by Edwige Cyffers (first semester) and Valentin Taillandier (second semester) The communication team was responsible for getting in contact with the IFÉ, developing the website in getting in touch on calculator community forums. Their activity is presented in part 5.

(b) Contribution to SageMath, led by Corentin Lunel This group focused on contributing to our initial pick for the computer algebra software, SageMath. This in an independent work that reflects our commitment to free and open software, and is detailed in part 6.

Part 2

Hardware

The hardware team is tasked with the design and production of the ***Symbo*****libre** calculator prototypes, following the guidelines and requirements issued by other groups; this includes planning the distribution of the product. One of the important goals is to write accessible and precise documentation for the assembly process, thereby supporting the *do-it-yourself* approach that we want to promote.

2.1 Hardware components

We carefully selected the components that we use for the calculator, to meet all requirements of affordability, availability, computational power and technical accessibility.

We have settled on the following list for the prototype that was presented in the project's public demonstration.

- The chosen core of the machine is a Raspberry Pi Zero.[1] Thanks to the Raspberry Pi Foundation, this computer offers a lot of processing power for a very low price of €5. It has a 1 GHz ARM core, 512 MiB RAM, and a GPU. It is also extensively tested and documented, making it a suitable platform for development and distribution.

  Due to the special business model behind the Pi Zero, it's actually difficult to order a large quantity for a unit price of €5; this will not be a problem to build a constant quantity of machines in the *do-it-yourself* approach.
- The keyboard has a custom layout and is printed on a specially-designed PCB, then wired directly to the Pi. The setup currently uses one wire per row and column, occupying a total of 14 pins on the Pi. The keys are 3D-printed.
- The prototype is powered by a 2000 mAh LiPoly battery, which is comparable in size and autonomy to the other battery-powered color graphing calculators.
- Along with the battery, we embark a charging and voltage boosting circuit, making it rechargeable through a common micro-USB cable; the PowerBoost 500[2] is our first choice. It is a pretty expensive board, but makes things simple for now as we want to avoid any powering issues. In the future we will look into replacing it or integrating it on the keyboard's PCB.
- Our current display is a 320×240, 2.4" LCD color screen.[3] It's connected to the Pi Zero through a 4-line SPI interface, which has the benefit of saving GPIO pins on the Pi for the keyboard.
- The operating system is stored on an SD card, currently 8 GiB. Any capacity of at least 1 GiB can be used.
- The case is 3D-printed.

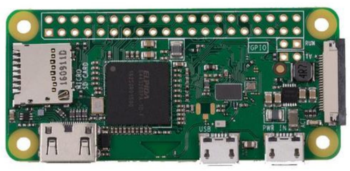
[1] `https://www.raspberrypi.org/products/raspberry-pi-zero/`
[2] `https://www.adafruit.com/product/1944`
[3] `https://www.buydisplay.com/default/qvga-2-4-inch-tft-lcd-touch-shield-320x240-serial-module-display-ili9341`

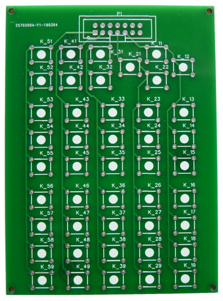
The total price of the components is detailed in Figure 2.1.

Component	**Price**
Raspberry Pi Zero	€5
PCB	€2
Battery	€11
PowerBoost 500	€13
Display	€7
Misc case, wires, solder	€5
Total	€53

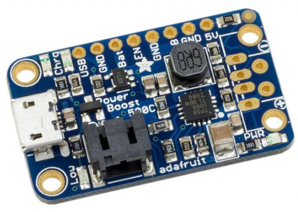
Figure 2.1: Prototype cost, excluding shipping.

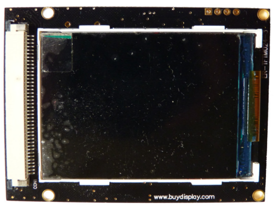
Some additional smaller components (such as an digital-to-analog converter to read the battery level, resistors or buttons) are required; we watched out for their availability and long-term support to make sure the *do-it-yourself* model remains viable in the future.


(a) Raspberry Pi Zero.


(b) Custom keyboard PCB.

(c) 2000 mAh LiPoly battery.


(d) Adafruit PowerBoost 500.


(e) EastRising 320×240 2.4" display.

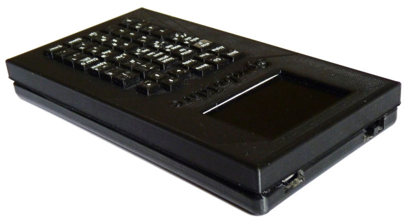

(f) Top side of the two-piece case.

Figure 2.2: Main calculator components.

We also ordered some elements directly from a Chinese supplier, to study alternatives to the previous list. This could lower the cost of the ***Symbo*****libre** calculator, although this study was not the main priority of our group during the semester, it is one the key points for the future of the calculator from the hardware team perspective.

During the project, we were also involved in the design and the development of specific hardware parts: the case and the PCB for the keyboard. Those parts are available in our code repositories in the form of source files, allowing anyone with the right tools (a 3D printer for the case, or tools and chemicals for the PCB) to reproduce our prototype, or to order the parts directly from suppliers that propose to produce parts from schematics. Both approaches are nowadays quite cheap, and we actually ordered five PCBs for just $2.

2.2 Current status

We managed to present a convincing prototype, meeting most of the requirements that we set as goals for the project. It is a true autonomous machine, with specifications that allows for typical real world use and realistic performance. The 45-buttons keyboard that we built was also fully working and handling various kinds of inputs. The prototype has a USB port that allows battery charging in a modern way, as well as an on-off switch to control the power.

2.3 Issues faced and future development

The Pi Zero has two USB ports, a standard one for powering and an OTG one for peripherals. We had trouble dealing with the latter; we wished to expose it on the case but we were not able to build a functional extension, likely due to OTG subtleties. So we could not communicate with a PC without opening the case and wiring directly on the Pi.

It also turned out that the assembly process, although involving only elementary techniques (soldering, gluing, 3D printing) was actually pretty tedious, and we think that for now the goal of easy distribution is not fully met. The keyboard requires a lot of soldering (4 Legs for each key), and a lot of wires are still lying tin the case, leading to bad connection issues.

To solve those issues, and now that we are more experienced with PCB design, we plan to implement some of the wiring onto the PCB, and thus have a minimal number wires and electronic boards inside the calculator.

One of the technical goal that we did not reach in time was to be able to monitor the battery level with precision (more than just the "low-battery" signal).

At the moment, since we still plan to enhance the prototype to solve those issues (although not critical, they are still important to us), we have not come up with clear and precise tutorials on how to build the prototype, since the prototype will significantly change in the short-term future work.

Part 3

Low-level aspects

3.1 System administration

This team is in charge of deploying a toolchain and a GNU/Linux operating system for the calculator. The OS comes with a computer algebra system, the Qt graphical library, and relevant drivers for the calculator hardware.

In essence, these tasks consist in configuring and building the kernel and packages, that are then installed on the Raspberry Pi's SD card. Because the Pi runs on an ARM processor, unlike typical computers that run on x86 processors, everything needs to be either compiled with an emulator (which is slow), cross-compiled (which is error-prone), or built on an ARM machine (which requires additional powerful hardware).

3.1.1 Choice of distribution

At first, our main concern about the choice of a GNU/Linux distribution was the integration of the computer algebra system SageMath[1], as it is already difficult to install it from source on `amd64` or `x86` architectures, and an early attempt at compiling SageMath directly on a Raspberry Pi had to be aborted after a week that had its share of system freezes (due to a lack of RAM) and compile errors.

The Gentoo distribution was finally chosen over Raspbian (a Debian derivative for Raspberry Pi), because in both cases a SageMath package would have had to be built from source, which is easier on Gentoo thanks to the sage-on-gentoo repository[2].

Another advantage of Gentoo in the case of Symbolibre is its great modularity. As the performance of the Raspberry Pi is limited, the system should be as minimalistic as possible, so the control Gentoo gives on compile-time options of packages can be useful. On the minus side, Gentoo can be a bit harder to use than other distributions and the compilation of the system takes some time.

When we moved away from SageMath to Giac, our needs evolved but we did not consider moving to another distribution because we already had a bootable Gentoo system with most of the needed software.

3.1.2 Creation of a bootable SD card image

We alternated between two methods to compile packages for the Pi's ARM target.

- Gentoo's package manager, `portage`, is able to cross-build packages for other architectures. This allowed us to compile packages from our personal computers.
- We emulated an ARM system on our computers, using the ARM emulator `qemu`[3].

The main inconvenience of the first method is that some packages do not support it, and compile errors are frequent. By contrast, the second method succeeds more frequently but takes around 20 times longer.

[1] We since then moved to Giac; see section 4.1.3 for more details.
[2] `https://github.com/cschwan/sage-on-gentoo`
[3] `https://www.qemu.org/`

We generally favored the first method, falling back to the second when we encountered errors that would have been hard to fix.

Our OS image includes:

- The Raspberry Pi's specific firmwares;
- A compiled Linux kernel;
- A collection of basic system utilities (`grep`, `find`,...);
- A graphical environment (see section 3.1.4);
- Our applications.

The image weighs less than 1 GiB, so it will fit on nearly any SD card. Documentation on how to create the image has been written for reproducibility purposes.

The system takes around 20 seconds to boot. Thanks to Plymouth, a widely-used splash screen manager that runs by default on many distributions, most of the logs are hidden. Unfortunately the experience is not yet flicker-free.

A tool for the user to install the image on an SD card was developed in collaboration with the system tools team, see section 3.2.3.

3.1.3 Device drivers

On a GNU/Linux system, the Linux kernel is responsible for communicating with the hardware, amongst other low-level tasks.

In order to save time, we used an already configured Linux 4.19 kernel from the Raspberry Pi Foundation[4]. This left us with two challenges to make the hardware fully functional: the display and the keyboard.

The controller of our display is an Ilitek ILI9341, which is common enough to have a stable driver in the Linux kernel tree: `tinydrm`[5]. Things go pretty smoothly once the driver is loaded and the screen is connected, except for GPU support which is a bit more involved.

As for the matrix keypad, it is a common thing to use on a Raspberry Pi, but the popular userspace approach is not suitable for the calculator. Instead the `matrix-keypad` kernel driver is used to drive the keyboard from kernel space, and integrates with the console and desktop environment. This way, the calculator's keypad is seen by applications as a regular keyboard.

3.1.4 Graphical environment

For the graphical environment that runs on the calculator, we are currently using sway, a light keyboard-oriented Wayland compositor. The GPU of the Raspberry Pi takes care of the rendering, and the frames are communicated to the SPI-connected display through PRIME offloading. Unfortunately, it makes the whole display refresh even if only a small region of the screen has been updated.

In the future, we could try to use the `eglfs` backend of the Qt graphical toolkit, without a wayland compositor, since all our applications are written with Qt.

3.2 System tools

The goal of this group is to develop interfaces between the calculator and its digital and human environment. We further specified our target by putting a strong emphasis on *the first contact.*

[4]https://github.com/raspberrypi/linux
[5]https://github.com/notro/tinydrm/wiki

We indeed believe the first contact is crucial for us. The calculator has to prove its good faith to the user that spends time to build it and maybe takes a risk by using it at school. Extra care is needed to compensate for the lack of any preceding reputation. This also implies that the calculator must be as intuitive as it gets. This group's tools are meant to contribute to this; they include the main menu, a setting manager, file transfers by using the calculator as a USB stick, and a desktop application to handle some non-trivial tasks on the calculator.

3.2.1 Main menu

The main screen is the program always available in background to start applications. It references all available applications, and has a simple select-launch logic.

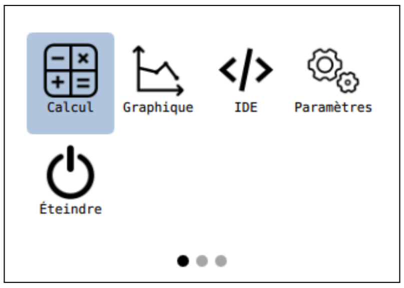


Figure 3.1: Main menu. Three dots show that tree panels are set up.

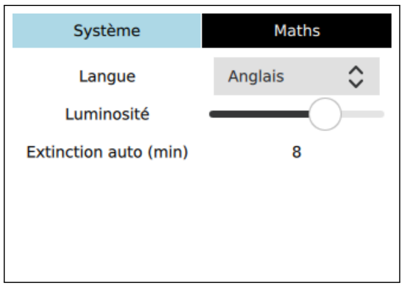
There are a lot of user input issues to take into account. The controls are a lot poorer than what a computer keyboard offers, so we must limit the amount of navigation in menus everywhere possible. Helped by the communication team, we decided on using a smartphone-like approach because the device shares common stakes with the calculator: a small screen and high navigational requirements. Typically this involves icons rather than text, which is now a common place for calculators to use.

Which applications appear in the menu is specified by the application developers and can easily be changed to account for user applications, special invocation conventions, or simply updates to names and icons.

3.2.2 Setting Manager

The same goes for managing application settings. On many calculators they are scattered in the applications and not all available at the same time, making it difficult to find and modify something specific. We chose to centralize all the settings in a single application, shown in Figure 3.2, with various types of settings and the associated widgets.

This application is a double benefit, as it also saves developers the need to design their own setting interface. Instead they can use the manager's simple API.



Langue Anglais
Luminosité
Extinction auto (min) 8

Figure 3.2: Setting manager.

3.2.3 Desktop application

Not all the users know how to burn an SD card, which is the first software step in the use of the calculator. Installing applications and updating the calculator are also non-trivial but necessary tasks. To make them more accessible, we developed a cross-platform, and of course free-software desktop application pictured in Figure 3.3.

The straightforward design idea is to have a button for each step and, activating them in sequence until the procedure is complete; the example shows the OS update process, after an update file has been selected and checked. Any error is shown in the status bar (hidden on Figure 3.3 to help troubleshoot issues. The OS install feature is currently only available on Linux, but porting it is not a major issue.

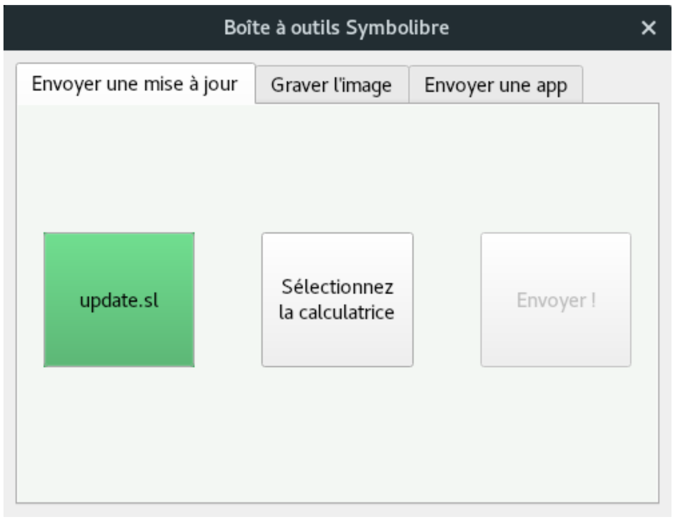


Figure 3.3: Desktop application, available in French and English.

We then extended the features of the application with an option to build the main menu entry of an application. As evoked in section 3.2.1, each application's metadata is stored in a configuration file with a fixed format. Developers can use this feature to create the configuration file for external applications. It is hidden by default.

3.2.4 File transfers through USB

The desktop application is only there for non-trivial tasks such as updating the operating system. File transfers can be done by simply connecting the calculator to an USB port; the calculator then behaves as a USB stick. This approach is becoming more and more popular with historical manufacturers and is the most straightforward way to go.

This features makes it even easier to transfer programs, text files or graph data from the calculator to analyze on the computer, or the other way round.

As we are aware some users will not want do download external software to transfer applications or updates to the calculator, we also made sure power users would be able to everything through the USB connection.

3.2.5 Further work

Not all the targets we had in mind could be developed in one year. Here is an overview of other tools we hope to develop in the future.

Terminal emulator We currently use the `QTermWidget`[6] library, but its integration in our applications can be refined with a bit of effort.

File Explorer This is the main shortcoming in our work. We lacked the time to develop this tool we deem mandatory. Currently the IDE accesses the filesystem with a custom widget, but there is no canonical way to do it, and we found no existing program that meets our requirements in terms of screen size. This should be the main target of development for the next version of the operating system.

Update and application finders Currently, the user has to manually download update files and applications, then use our desktop application to install them on the calculator. We can imagine using the desktop application to search and download software updates. In the same category, we can add to our website a section with ready-to-use external applications.

[6]https://github.com/lxqt/qtermwidget

Part 4

Main applications

4.1 Math applications

Most of the working groups are concerned with building the ***Symbo*****libre** calculator as a computer. This group focuses on what makes it a graphing calculator, the math applications: its calculator, function plots, equation solvers among others.

All rely on the same math computation core and its interfaces, which must be carefully designed to be modular and easily usable. As we had to lower our expectations over time, we focused on building this core and the most fundamental applications instead of a complete suite.

4.1.1 Current state

The core of the math applications currently consists of three reusable components that are available to link against as C++ libraries.

- A *pretty printing* system to display and interactively edit math formulas printed in 2D notation, called *Edition Trees*;
- A layer of abstraction and communication with Giac, called *SLL*;
- A plot widget with a rich interface to build graphs.

The widget and the applications are developed using Qt Quick, which is a modern toolkit but has a somewhat steep learning curve. The calculator also includes these applications:

- An interactive calculator with formal computations and numerical approximations;
- An application to define and plot functions.

4.1.2 Calculator application

The calculator is shown on Figure 4.1, with the virtual keyboard used for debugging, the outline of the program's structure is presented on Figure 4.2. Notice how Edition Trees and SLL are used together.

Pretty printing both the input and output of the calculator is essential to provide visual feedback. The calculator is thus able to show and edit arbitrary expressions in real time, including expressions that are not yet meaningful or well-parenthesized. This is the role of Edition Trees, basically analyzing enough structure to display while leaving enough freedom to perform edition in near-constant time. Details are provided in section 4.1.4.

When the user validates the input, the edition tree generates a string for to the formula and sends it to SLL, a dedicated calculation interface described below. SLL sends back the result which is then displayed as an Edition Tree.

4.1.3 The core of the computation: SLL

Giac as the formal calculator

Implementing a halfway-decent exact computation system was always out of the question; from the start we decided to use an open-source computer algebra system. We currently use Bernard

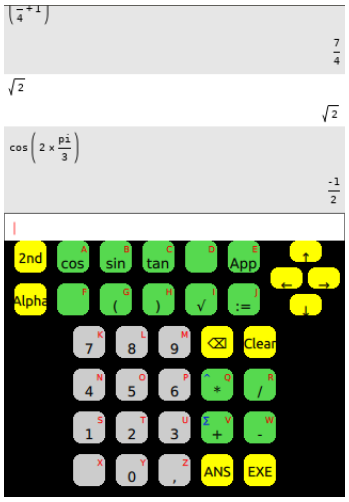


Figure 4.1: Calculator screen

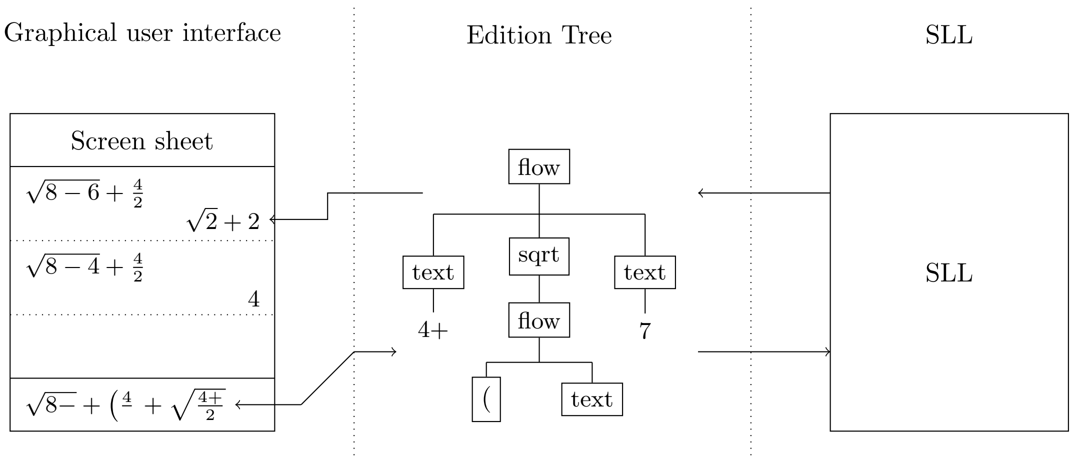


Figure 4.2: Structure of the calculator

Parisse's Giac [6]. As the CAS of the HP Prime, we had an upright guarantee that it provides all the functions we need for a reasonable impact on the hardware. Even though high schoolers are generally not interested in computer algebra, having powerful exact calculus and equation solving within reach justifies in itself the use of Giac.

Choosing the CAS was not immediate. We initially had a try at SageMath, which is a very powerful, community-maintained interface to a variety of computer algebra software, including Giac. Our attempts to trim it down to fit in the performance range of the Raspberry Pi Zero were however unsuccessful, so we turned to our current solution Giac.

Performance	SageMath	Giac
Launching time (s)	115	1.2
100 thousand decimal of $\sqrt{2}$ (s)	8	15
Memory at startup (MiB)	175	13

Figure 4.3: Quick resource comparison of GIAC and SAGEMATH on the Pi Zero.

SLL: the calculation interface

SLL is an abstraction layer between the applications and the CAS. It provides an interface independent from the underlying tool (currently Giac), as well as a custom syntax and set of functions. The name stands for *"SymboLibre Language"* and expresses the idea that the mathematical language of the calculator might not be exactly that of the CAS.

SLL does not perform any computation, but communicates expressions to the formal calculator in a suitable syntax. It consists of a parser to transform the text and a post-converter that transforms Giac's result into an Edition Tree.

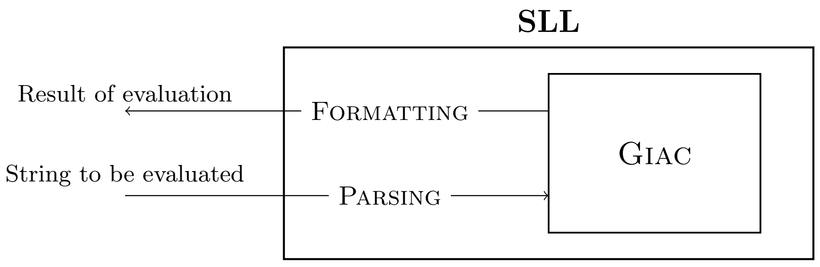
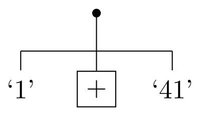


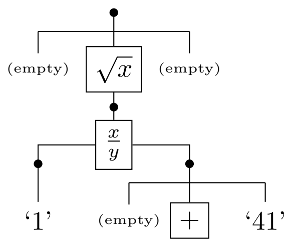
Figure 4.4: Structure of SLL

4.1.4 Edition trees

Edition trees are a structure thought to manage the quick edition and rendering of mathematical expressions. It is essentially a tree whose levels alternate between two types of nodes: *flows* that represent the horizontal elements of the formula put together as a string, and *semantic nodes* that have mathematical meaning and a 2D representation. Leaves are either semantic nodes without children or raw text. In the examples below, flows are drawn as • and semantic nodes as squares.



(a) Edition Tree for $1 + 41$.



(b) (Simplified) Edition Tree for $\sqrt{\frac{1}{+41}}$

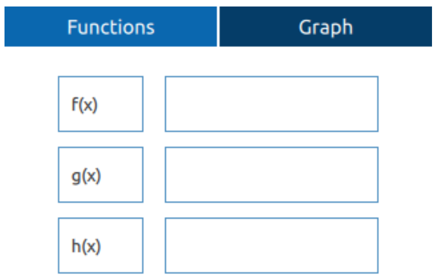
Each edition tree also keeps track of the *cursor*. Upon validation, the edition tree generates its expression in a textual form that is then sent to SLL for parsing and evaluation.

This apparently simple object turns out to be quite deep. It requires a subtle balance between parsing to display, and keeping text to edit. When parsing too much, edition operations increase in implementation and algorithmic complexity. When not parsing enough, the alignment of certain natural elements such as exponents cannot be guaranteed.

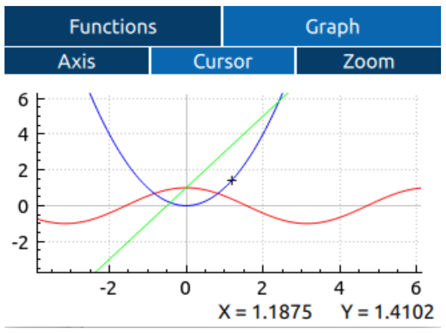
The conversion from edition trees to text is straightforward, but pretty printing Giac objects is another challenge. This conversion dives into Giac's internal representation of objects to produce an edition tree.

4.1.5 Function application

This applications can be used to define and plot functions in an interactive graph. It's built using the declarative Qt Quick paradigm but integrates the graph widget which is implemented in the Qt Widgets C++ framework. The widget is discussed in section 4.1.6.



(a) Function application home screen.



(b) Function application graph screen.

The architectures of the applications are written for the most part in a declarative language called QML. QML defines window elements in a object-oriented manner and provides means of communicating between them. The most important ones are *signals* and *property binding*.

An important QML property is called the *focus*. The object that possesses the *focus* will capture the input of the user. Most of the work on this interface was directed to properly distribute and move the focus according to the user input. It required to develop a whole set of dedicated objects.

The function graph is itself a QML object based on the *Graph Plotter*, our low level tool responsible for plotting curves. It gives a layer of QML communication between the Graph Plotter and the rest, that is to say data coming from another part of the application, or even inputs.

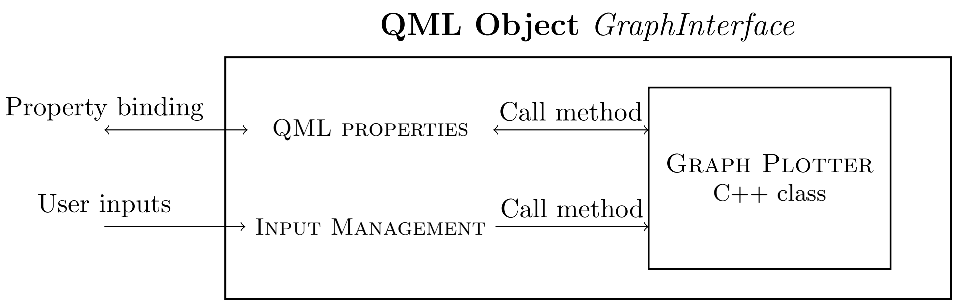


Figure 4.7: QML interface of the graph plotter.

4.1.6 Graph plotter

The Graph Plotter consists in a standalone library, making it easy to link in other applications as shown in the previous section. The main goal is to plot function graphs and implement graphical tools such as a cursor following the curves, finding extrema, evaluating surfaces or finding intersection points.

Rather than trying to re-develop everything from scratch, we felt more confident in adapting an existing solution to address our needs. Qt does not provide directly a complete and out-of-the-box library to render graphs, but it turns out there exists a library that covers all our needs, called `QCustomPlot`. There is one downside that hindered our progress at first. It is designed to work with QtWidgets, the classical Qt framework for rendering widgets, while we want to use the more modern Qt Quick framework. Fortunately workarounds are possible and have already been implemented.

For now, the Graph Plotter can plot multiple curves, with a beginning of discontinuity management, that is still very perfectible. The use of `QCustomPlot` really helps by taking care of most of the technical rendering issues. Furthermore, the library offers a fluid navigation, through the use of either a free or curve-bound cursor. The movement of the window with the cursor only require a local update of the curves, without the need to recompute the function on the whole domain. On the other hand, the zoom requires a complete update of all the curves. However, the cost of this computation is no issue for the Raspberry Pi Zero.

Finally, the API developed for the library remains pretty simple and basic. It allows no direct communication with `QCustomPlot`, reducing the risks of misuses. An internal, supplementary layer ensures the communication between the API functions and the `QCustomPlot` object used internally to make all the renderings.

4.1.7 Conclusion and Future Work

Programming a full fledged calculator by the end of the year was too ambitious, and we could not reach our goals. However, we laid a solid foundation for the Symbolibre. The calculator and the function application show how we successfully assembled our different tools, which makes us quite optimistic for future improvements.

Some ideas of future work:

1. Improve the features of the calculator: scrolling, more user friendly rounding, type previous expression, etc;
2. Improve the flexibility of the function application: arbitrary number of defined functions, change names, customization options;
3. Improve the Graph Plotter analysis tools: find discontinuity points, automatic focus on relevant parts of the curve;
4. Improve the SLL interface;
5. Develop more applications on the basis of the function application, for probabilities, statistics, etc.

4.2 Programming and the IDE

The programming team was divided in two independent groups. The first one worked on the text editor, which will become a full-fledged IDE in the future. The second one developed TI-Basic and Casio Basic interpreters, adding to the base support for Python and OCaml that are installed at the operating system level.

4.2.1 Language choices

We decided to support both TI-Basic and Casio Basic to improve the compatibility of our calculator with the existing pedagogical tools. We are aware that introducing a new language in a full-TI of full-Casio high-school class is deemed to fail.

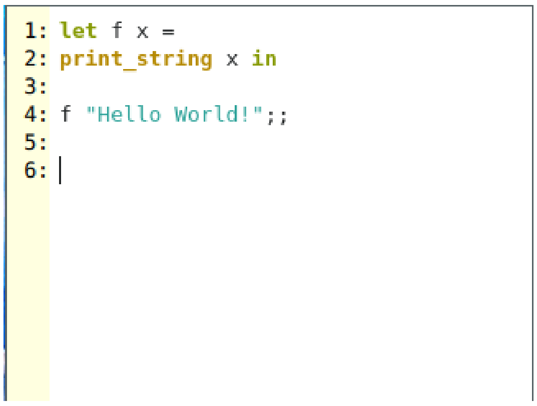
This is to fit the current habits of teachers and pupils. TI-Basic and Casio Basic are widespread in high-school to teach programming and computer science. Therefore maths books only provide examples for these languages; we must support them if the calculator is to be used at any degree.

Common languages such as Python and OCaml are must-haves: both are officially present in teaching programs in France. Python is currently introduced in *seconde* and OCaml in scientific *classes préparatoires*. A good level of compatibility will allow students to develop programs on a computer and test them on the calculator, or the other way round, making the calculator a suitable tool whenever no computer is available.

4.2.2 Text editor

The text editor on top of the Qt documentation, that provides a very good basis with little effort. It is extended with the KDE `syntax-highlighting` library[1] which supports a wide variety of languages, including TI-Basic, Python and OCaml. This simple combination makes for a decent syntax-highlighted text editor with large opportunities of extensions and reasonable coding effort.

As the screen is small, we decided to favor readability and space by reducing the amount of controls on the screen, instead using the keyboard to access most features.

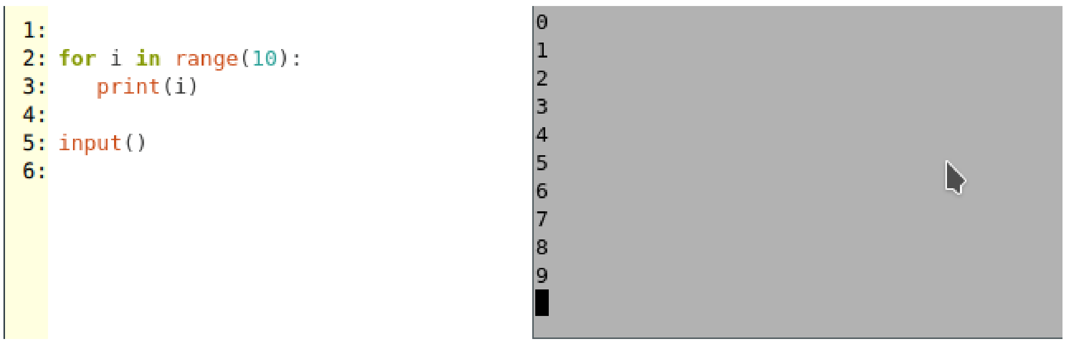


```
let f x =
print_string x in

f "Hello World!";;

```

Figure 4.8: OCaml Example



```

for i in range(10):
    print(i)

input()

```

```
0
1
2
3
4
5
6
7
8
9
```

Figure 4.9: Python Example

Controls

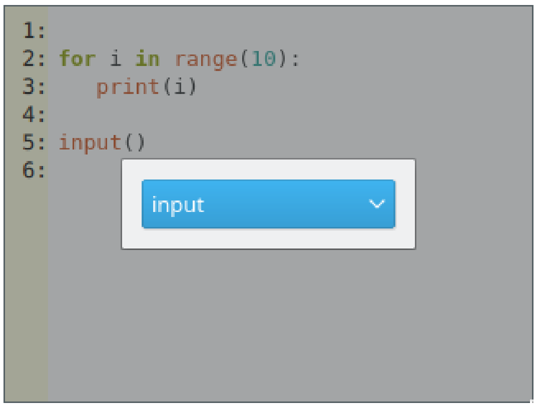
It is commonly agreed that writing code on a calculator is difficult due to the layout of the keyboard, and sometimes even hindered by impractical user interfaces. The calculator keyboard has a row with keys dedicated to programming structures, that can be dynamically bound to frequently used snippets such as control statements and loops.

The modularity of the IDE allows the user to create additional language-dependent snippets or rebind the programming keys of the keyboard to use their own. The default setup provides control structure and common input/output functions, and is shown in Figure 4.10.

From a text editor to an IDE

In order to make programming as straightforward as possible, the text editor integrates an interface to a terminal emulator to execute programs, using the `QTermWidget` library.[2] A suitable

[1] https://github.com/KDE/syntax-highlighting
[2] https://github.com/lxqt/qtermwidget



```

for i in range(10):
    print(i)

input()

```

Figure 4.10: Snippet insertion

interpreter is chosen based on the file's extension, making the feature once more extensible to new programming languages.

4.2.3 Basic languages interpreter

Along with Python and OCaml, we decided to provide support the algorithmic subsets of TI-Basic and Casio Basic, as they are still used by many teachers. Since there are no open-source implementations, the operating systems ships custom interpreters.

Both languages have a tremendous amount of functions, but only the basic commands are implemented. Here is an overview of the capabilities of the interpreters:

- Expressions and arithmetic.
- Strings, with most of the usual operations.
- Emulation of a Casio calculator's global memory. All variables on these models are global, which turns out to be important for some math exercises using several programs.
- A partial support of lists and matrices.
- Miscellaneous and utility functions.
- Usable error diagnoses, in an attempt to improve on the existing poor messages provided by Casio and TI calculators.

Part 5

Communication

This team handles the communication around the project at various levels. This includes the the development of our website and our relationship with the *Institut Français de l'Éducation*[1] (IFÉ), but also the coordination of the other groups. We also had chances to interact with end users: students as well as teachers.

5.1 Visual identity

We deemed the visual impact of a logo important, so we designed several versions based on a dove to remind of the free nature of the project. The end result is the dove carrying a calculator represented on the left on Figure 5.1.

It quickly became apparent the logo needed to be simplified to better render on the low-resolution screen of the calculator, or as an icon of the desktop application. We thus designed a flat version, shown on the right.

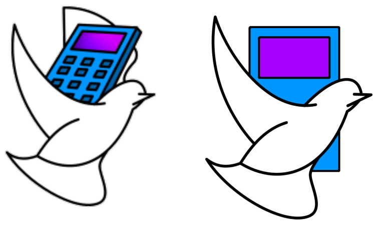


Figure 5.1: The ***Symbo*****libre** dove and its flat version.

A proper visual identity requires the applications to follow the same color and font schemes. Though we lacked the time to implement a common style, we spent some time designing guidelines for developers to follow to improve the usability of their applications. This includes:

- Using a common color scheme;
- Using fonts from a selected list, namely *Accanthis ADF* for the infography and *DejaVu Sans Mono 11* for common text. The small size of the screen and the rendering algorithms put constraints on these choices.
- A set of key binding principles;
- Compliance with the layout of the main menu.

Figure 5.2: *Accanthis ADF Std Bold* used in the full project logo.

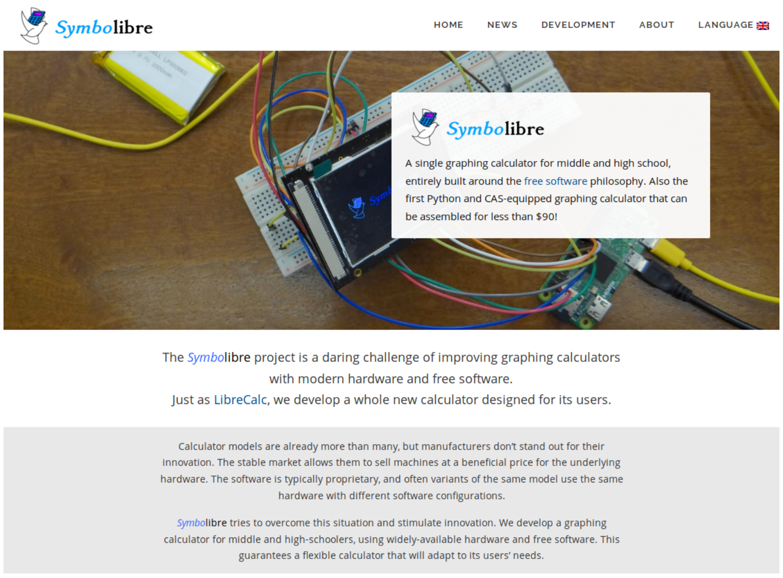
[1]http://ife.ens-lyon.fr/ife

5.2 External communication

We managed external communication on multiple fronts at the time. We designed a website, we printed flyers, we made contact with the IFÉ and also tried to identify the needs of our end users through a survey.

5.2.1 Website

The ***Symbo*****libre** project's website at `https://symbolibre.org` was developed to present our work from the net, to partners as well as calculator communities. Much like this report, it includes a presentation of the motivations, the calculator and the team.



*Symbo*libre

A single graphing calculator for middle and high school, entirely built around the free software philosophy. Also the first Python and CAS-equipped graphing calculator that can be assembled for less than $90!

The *Symbo*libre project is a daring challenge of improving graphing calculators with modern hardware and free software.
Just as LibreCalc, we develop a whole new calculator designed for its users.

Calculator models are already more than many, but manufacturers don't stand out for their innovation. The stable market allows them to sell machines at a beneficial price for the underlying hardware. The software is typically proprietary, and often variants of the same model use the same hardware with different software configurations.

*Symbo*libre tries to overcome this situation and stimulate innovation. We develop a graphing calculator for middle and high-schoolers, using widely-available hardware and free software. This guarantees a flexible calculator that will adapt to its users' needs.

Figure 5.3: The homepage of the website.

5.2.2 Flyer

Apart from the website, we promoted the project with the flyer you may see in figure 5.4. Those were printed and scattered all around the school before the public demonstration.

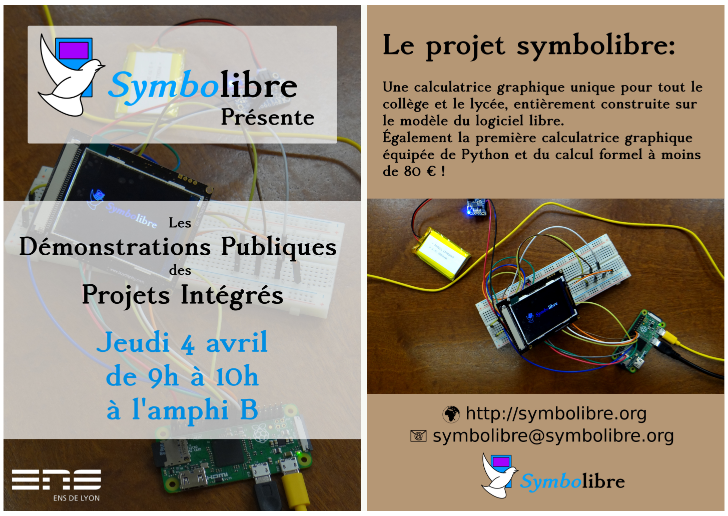
5.2.3 Partnership with the IFÉ

As we mentioned in the introduction of this report, we tied strong links with teachers from the IFÉ during the development of the project. Their experience helped us nail the needs of both teachers and students in terms of user experience, and directed many choices in the design of the applications.

We met them several times to present the project and the prototype. At the public demonstration, students from the Lycée Albert Camus came to try out the calculator, providing a vast amount of feedback in a short amount of time.

5.2.4 Survey

As part of the development of our calculator, a poll has been set up for high school students and shared with a few math teachers around Lyon. The goal was to determine the common problems met by students when using their calculators. We got more than a hundred answers, so we can draw pretty reliable conclusions on the challenges the ***Symbo*****libre** calculator has to address. Some results are shown in Figure 5.6.


Symbolibre
Présente
Les
Démonstrations Publiques
des
Projets Intégrés
Jeudi 4 avril
de 9h à 10h
à l'amphi B
ENS DE LYON

Le projet symbolibre:

Une calculatrice graphique unique pour tout le collège et le lycée, entièrement construite sur le modèle du logiciel libre.
Également la première calculatrice graphique équipée de Python et du calcul formel à moins de 80 € !

http://symbolibre.org
symbolibre@symbolibre.org
Symbolibre

Figure 5.4: The two faces of the flyer.

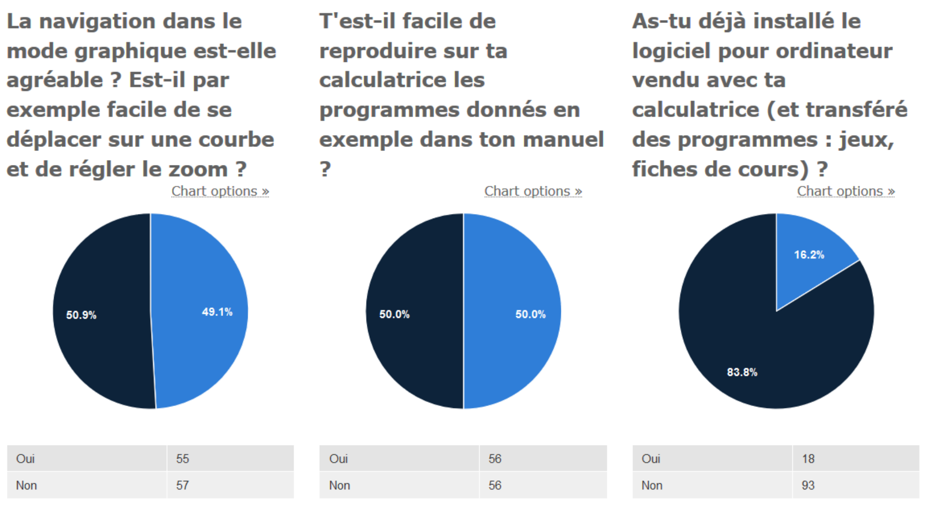
Figure 5.5: Meeting with teachers from the IFÉ.

And the first conclusion we have drawn from it is that few good points in their calculators are raised by the students. Apart from having a readable display, none of the aspects we suggested reach general satisfaction. This led us to determine which improvements the Symbolibre calculator should bring, which are notably in the clearness of its settings, in the use of the graph application and programming environment, in the user manual, in a straightforward keyboard and, last but not least, in the price.

5.2.5 Discussions with Allizé plasturgie

The first ***Symbo*****libre** prototype is 3D-printed. This technology makes it very easy to try out several designs in a short amount of time, but needs about 12 hours to print all the pieces of the case and the keys. We had an opportunity to get in touch with Allizé Plasturgie, the Lyon plastics union, to discuss of other ways of producing these components. The chief representative of the union expressed interest in working together with ***Symbo*****libre** to guide us on the production process. This partnership is a fine opportunity for the project.

Since the union also manages training centers, the design of alternative cases might be a practical



La navigation dans le mode graphique est-elle agréable ? Est-il par exemple facile de se déplacer sur une courbe et de régler le zoom ?

Oui	55
Non	57

T'est-il facile de reproduire sur ta calculatrice les programmes donnés en exemple dans ton manuel ?

Oui	56
Non	56

As-tu déjà installé le logiciel pour ordinateur vendu avec ta calculatrice (et transféré des programmes : jeux, fiches de cours) ?

Oui	18
Non	93

Figure 5.6: Some answers to the online poll.

exercises for real-life students. This way the project stays a project by student, for students, with students. Transdisciplinarity is the key.

5.2.6 Future concerns about the examination mode

A few years back, the French government initiated a law process to formalize and further secure the usage of calculators during examination. Among other things, it requires calculator to have a special *examination mode* with a blinking LED, locked communications and unaccessible memory until the calculator is plugged into a computer.

This legislation is not implemented yet, but we must plan a way to support it before the ***Symbo*****libre** calculator can be of any use to students. Only very few resources are available, the main one being the official journal that introduces the regulation. [1]. Just as others students or even teachers of the IFÉ, we are left with a vague description that no two manufacturers implement in the same way.

Part 6

Contribution to SageMath

This team aims at bringing some contribution to the computer algebra system SageMath. It was indeed our first pick for the calculator. Following to the free software spirit of the project, some of us decided to contribute back to this software. When SageMath was replaced by Giac in the calculator, the contribution was advanced enough so we went on with it.

A review of existing material in SageMath led us to work on Turing machines as an extension of the many tools available on finite automata in SageMath; this is a topic all of us are familiar with thanks to our theoretical computer science background.

6.1 Work outline

We quickly decided to split the work into two groups.

- The first one took over bibliographic research to find an efficient encoding and simulation that would match SageMath's requirements.
- The second one explored SageMath's tutorials and documentation, summing it up for the other members of the project.

Once we had enough insight on how to integrate a contribution in SageMath's codebase, we started working on three general goals:

- The implementation of Turing machines and their simulation, which is currently written as an external program using SageMath;
- The graphical representation of the Turing machines, which involves graph representations and planar graphs in particular;
- Communicating with the SageMath community to discuss the integration of our work in the library. Members of the two teams have gotten in touch.

6.2 Bibliography work

The first and most important resource available is SageMath's documentation.[1]

- *"Welcome to the Sage Developer's Guide!"* describes the requirements of the SageMath community in terms of coding style, how to properly use tools and how to document code. An interesting aspect is that every SageMath object must have a LaTeX representation, which we support as well.
- *"Finite State Machines, Automata, Transducers"* is the documentation about finite states machines. Our library will extend the current module where automata are implemented.

Summaries of both documents have been distributed to the team and saved to the repository wiki. They allow us to draw the lines of the second part of the project.

[1] https://doc.sagemath.org/html/

6.3 Tutorials

None of us had a deep prior experience with SageMath, so the second group learned the use and structure of SageMath and produced tutorials for the rest of the team, covering:

1. Install SageMath from binaries and building from source code
2. Basic aspects, types and help using the interactive shell
3. Defining and using functions
4. Solving equations and systems of equations
5. Differentiation, integration and differential equations

These tutorials are designed for users who are already familiar with Python. Consequently, they focus on the differences between SageMath and Python. We try to give an insight on some fundamental SageMath libraries that may be integrated to the calculator.

6.4 Turing machines implementation

We chose to be as close as possible to the code of the finite state machines as Turing machines are natural extension of such machines. This implies using SageMath's object-oriented model. The main finite state machine class is flexible enough to represent finite automata, transducers and Turing machines. However the classes defined for the transitions, the tapes and the processors are not fit for Turing machines. The head cannot go back in the tape because the other automata's only mechanism is moving forward.

We chose to keep the automaton and states, but defined a new transition class allowing moving reading head and infinite tape has been written, shown in Figure 6.1.

```
sage: t = TMTransition('A','B',[0,1,1],[1,0,1],[1,-1,0],3)
sage: t
Transition from 'A' to 'B':
  0|1,>
  1|0,<
  1|1,-
```

Figure 6.1: Representation of a transition for a Turing machine.

The infinite tape is represented by a doubly linked list, which perfectly matches the space requirements as they can be arbitrarily enlarged in both directions while staying finite. Support for Turing Machines with multiple tapes is also implemented.

SageMath usually provides versatile functions in terms of typing and conversion, and tends to catch typing errors. An example of this type of mechanism is shown in Figure 6.2, where a Turing Machine is defined from a set of transitions, and the states are inferred.

Some work still needs to be done here. We already have the theoretical algorithms to run machines (the heuristics), but their integration with the automaton simulation classes of SageMath raises issues with multi-tape situations. A new way of managing multi-tape machines is probably required to properlys fit in the existing codebase.

6.5 Turing machines graphical representation

Producing a readable and potentially beautiful representation for Turing machines was a major point of our work. We studied several options.

```
sage: T = TuringMachine(3,[t])
sage: T
Turing Machine with 2 states and 3 tapes
sage: T.states()
['A', 'B']
```

Figure 6.2: Turing machine built from the previous transition.

Our first take was using pre-existing SageMath functions to output the LaTeX representation of an automaton, that could be adapted to display edge labels. However, the vertices are only arranged in regular polygons as in Figure 6.3. This option is too simple for anything larger than four vertices so we moved on to other methods.

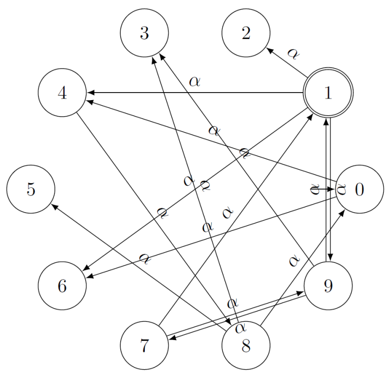


Figure 6.3: SageMath's default automaton layout.

The next option we considered was to use SageMath functions for graphs. They have more sophisticated algorithms to arrange vertices, but call back to a LaTeX library that cannot represent symbols such as initial and final states.

So we decided to print Turing Machine automatons as *trees*. According to our experience, this type of layout matches best with the way we read them. Our algorithm produces a tree rooted at the initial state, were the children of each state are the states they transition into but are not already shown. See Figure 6.4 for an example.

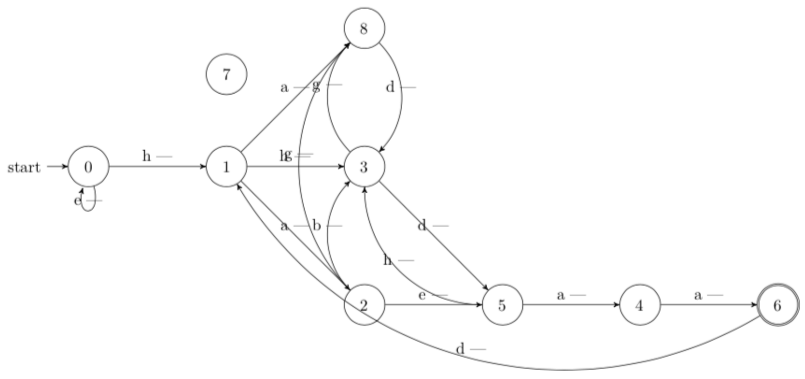


Figure 6.4: Tree representation of a Turing Machine.

This tree structure highlights the computation flow from left right, making it easier to understand the automaton. In the same time, it reduces the number of possibly crossing arrows. This property works well for most Turing machines according to our experiments.

There are not a lot of options for the tape; we print it as a grid with an arrow indicating the current position of the head.

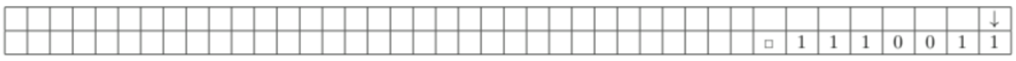


Figure 6.5: Tape representation.

SageMath has many graph libraries, so we thought using them to compute a representation minimizing the number of intersection between edges. Unfortunately this algorithm is not of polynomial complexity, making it long to show large graphs; and the planar arrangement is not necessarily the most meaningful one.

6.6 Future improvements

We had to extend SageMath's graph representations to display edge labels. There as still display issues such as edges can crossing labels; they can be addressed by computing positions of labels and paths of edges. We are studying this option.

The methods that will make the Turing machine class callable to decide whether a word is in a language are not yet complete. This a more of a software composition issue than an algorithmic one, as the algorithms are already chosen.

Finally, we got answers from the SageMath developer community forum. By staying in contact with them we hope to integrate our library in the SageMath upstream.

Part 7
Acknowledgements

This project was an immense machinery involving close to 35 people over the course of the year.

We would like to thank the Computer science departement at the ENS de Lyon, which funded the hardware for the prototypes and the website's domain, and the Avalon team, which hosts the website and provided some Raspberry Pis for us to use.

Thanks to Eddy Caron for supervising the project during the whole year and providing us with well-needed (and sometimes painful) insights on the art of managing a large project.

Our special thanks also go to Corinne Raffin and Gilles Aldon of the IFÉ, for putting so much interest in the project from the very beginning. Meeting you was a motivation boost just as much as a relevant opportunity to improve the project.

Thanks also to the students of the Lycée Albert Camus who came to test our prototype on the day of the public demonstration. We had a great time going over it with you, and we hope that we can do this again sometime.

Finally, thanks to Xavier Andréani for testing the machine and writing a positive review that triggered a lot of reactions on TI-Planet [3], to Théophane Gros for helping us setup the display and understand more of the electronics, and to Bernard Parisse for providing us with quick and accurate support on all things Giac.

Part 8
Conclusion

Over the course of the year, we achieved the main goal of the project and managed to build a working graphing calculator, despite having to regularly review our schedules and objectives on the software aspects.

The academic year has now ended, but the ***Symbo*****libre** project has not. Many of us would like to take the project further into development.

Our first prerequisite is to make the source code available to everyone online, under a GPLv3 license. There are no major barriers to that, it is mostly a question a sorting out everyone's agreement and license headers.

Another short-term goal is to review and harmonize the code written by different people. The high concentration of contributions the last few weeks made it too hard to review code as it was written.

As for our long-term plans, we are discussing the possibility of testing some of our prototypes with high school students, an opportunity made possible by our partnership with the Institut Français de l'Éducation. Other people have expressed interest in the project at the public demonstration and on community forums, and we are looking forward to opportunities to work with them.

Bibliography

[1] Élaboration des projets du programme du nouveau lycée. Mathématiques, page 14. `http://cache.media.education.gouv.fr/file/CSP/19/8/2de_Mathematiques_Enseignement_commun_1021198.pdf`.

[2] Xavier Andréani. Championnat des Pythons : performances. `https://tiplanet.org/forum/viewtopic.php?f=49&t=21514`.

[3] Xavier Andréani. Symbolibre: calc française Python et formelle (giac de Xcas). `https://tiplanet.org/forum/viewtopic.php?f=112&t=22470`.

[4] Romain Goyet et al. Epsilon – Modern graphing calculator operating system. `https://github.com/numworks/epsilon`.

[5] Pierre Parent and Ael Gain. LibreCalc (archive). `https://web.archive.org/web/*/librecalc.com`.

[6] Bernard Parisse. χCAS, the swiss knife for mathematics. `https://www-fourier.ujf-grenoble.fr/~parisse/giac.html`.